The Normalized Radial Basis Function Neural Network

Felix Heimes
Lockheed Martin Control Systems
Johnson City, NY 13760

Bram van Heuveln
Department of Philosophy
Binghamton University
Binghamton, NY 13902

ABSTRACT

This paper presents a new neural network called the Normalized Radial Basis Function (NRBF) neural network. The NRBF integrates techniques from two similar neural networks: the General Regression Neural Network (GRNN) and the Radial Basis Function (RBF) neural network. The NRBF is identical to the standard Radial Basis Function (RBF) network except the hidden layer outputs are normalized before being passed through the output layer. The normalization of the hidden layer weights is shown to improve the extrapolation performance of the conventional RBF network. We have reason to believe that under normal circumstances the NRBF outperforms the RBF and the GRNN..

1. INTRODUCTION

The Radial Basis Function (RBF) neural network was proposed by Broomhead and Lowe [3]. This neural network is very different from neural networks with sigmoidal activation functions in that it utilizes basis functions in the hidden layer that are locally responsive to input stimulus. These hidden nodes are usually implemented using a Gaussian kernel function. The RBF training procedure requires that a clustering algorithm determine the location (center) of each Gaussian kernel and the width of each kernel. Once the hidden layer nodes are selected the output layer weights can be determined analytically using a Least Mean Squares (LMS) method which determines the optimum output layer weights based on the training data and the selected hidden node parameters.

Another similar neural network, the General Regression Neural Network (GRNN) was introduced by Donald Specht[1]. The GRNN was introduced as a memory based neural network that would store all the independent and dependent training data available for a particular mapping. When presented with a new input vector, the GRNN uses a distance function to compute the weighted average of the dependent training data whose independent parameters are in close proximity to the input vector.

Apart from the fact that the GRNN and the RBF were motivated from different principals, their implementation is very similar. Both networks perform identical processing in the hidden layer and both networks perform linear operations in the output layer. The main difference is that the GRNN output layer performs a weighted average while the RBF performs a weighted sum.

Which of these two networks is better depends mainly on the application. The GRNN definition requires that it store the majority of the training data. When computational constraints are not significant, such as in stock market prediction where predictions may only be required once per day, it is probably best to use the memory based GRNN. However in most other applications, such as embedded control systems, computational efficiency is more crucial and the RBF network would be preferred.

This paper will study the effects of the GRNN and RBF training methods to gain a better understanding of each networks strengths and weaknesses. This paper will also integrate GRNN and RBF differences to improve the capabilities of both networks. Specifically, we will investigate combining the normalization of output layer weights performed by the GRNN with the pseudo-inverse method for setting the output layer weights of the RBF. This new idea presents an extension to the definition of the RBF network. We will call this new neural network the Normalized Radial Basis Function (NRBF) neural network. We will demonstrate that the NRBF presents qualities that are an improvement of both the GRNN and the RBF with a simple example.

2. RADIAL BASIS FUNCTION DEFINITION

The RBF neural network is described by Broomhead and Lowe[3]. The hidden layer of the RBF compares a new input vector (x) with a number of stored pattern vectors (c) based on an arbitrary distance function $\|x-c\|$. Usually the distance function is the Euclidean norm with an additional normalization constant in each dimension:

$$\|x-c\| = \sqrt{\sum_{k=1}^{n}\left(\frac{x_k - c_k}{\sigma_k}\right)^2} \qquad (1)$$

where x and c are both vectors of length n and σ_k is the normalization constant that controls the width of the basis function. A clustering algorithm is normally used to define the number, location, and widths of the centers of the basis functions. It is obvious that the more basis functions that are included in the network the better its accuracy. However, the number of basis functions normally needs to be limited for practical reasons.

This distance measure is then used to evaluate the basis functions: $\phi(\| x-c\|)$. If one uses a Gaussian basis function the hidden layer outputs are evaluated as follows:

$$\phi(\|x-c\|) = e^{-\|x-c\|^2} \qquad (2)$$

Given a network with p basis functions, a predicted scalar output y_j', is computed using the following equation:

$$y'_j = \sum_{i=1}^{p} \lambda_{ij}\phi(\|x - c_i\|) \qquad (3)$$

Since the output layer equation is a pure linear transformation, the coefficients λ_{ij} are analytically determined using the LMS algorithm. Consider matrix A to be the q-by-p matrix, where:

$$A_{li} = \phi(\|x_l - c_i\|) \qquad l = 1,2,..., q, i = 1,2,.. p \qquad (4)$$

In Equation (4), each x_l is one of q training vectors for which the actual output is known to be an m element vector y_l. Using A, we can express the computation of the RBF output layer as a matrix multiplication:

$$y' = A\lambda \tag{5}$$

where λ is a *p-by-m* matrix of weights, A is a *q-by-p* matrix of hidden layer outputs for all training patterns, and y' is a *q-by-m* matrix of network outputs. Since ideally the prediction equals the desired output:

$$y' = y \tag{6}$$

we can get an optimum prediction based on the training data by setting the matrix of weights (λ) according to:

$$\lambda = A^{+} y \tag{7}$$

In Equation (7), A^{+} is either the inverse if p = q, or the pseudo-inverse if p < q.

3. GENERAL REGRESSION NEURAL NETWORK

The GRNN predicts the value of one or more dependent variables, given the value of one or more independent variables. The GRNN thus takes as an input vector x of length n and generates an output vector (or scalar) y' of length m, where y' is the prediction of the actual output y. The GRNN does this by comparing a new input pattern x with a set of p stored patterns x^i (pattern nodes) for which the actual output y_i is known. The predicted output y' is the weighted average of all these associated stored outputs y_{ij}. Equation (8) expresses how each predicted output component y'_j is a function of the corresponding output components y_j associated with each stored pattern x^i. The weight $W(x, x^i)$ reflects the contribution of each known output y_i to the predicted output. It is a measure of the similarity of each pattern node with the input pattern.

$$y'_j = \frac{N_j}{D} = \frac{\sum_{i=1}^{p} y_{ij} W(x, x^i)}{\sum_{i=1}^{p} W(x, x^i)} \qquad j = 1, 2, \ldots, m \tag{8}$$

It is clear from Equation (8) that the predicted output magnitude will always lie between the minimum and maximum magnitude of the desired outputs (y_{ij}) associated with the stored patterns (since $0 \le W \le 1$). The GRNN is best seen as an interpolator, which interpolates between the desired outputs of pattern layer nodes that are located near the input vector (or scalar) in the input space.

A standard way to define the similarity function W, is to base it on a distance function, $D(x_1, x_2)$, that gives a measure of the distance or dissimilarity between two patterns x_1 and x_2. The desired property of the weight function $W(x, x^i)$ is that its magnitude for a stored pattern x^i be inversely proportional to its distance from the input pattern x (if the distance is zero the weight is a maximum of unity). The standard distance and weight functions are given by the following two equations, respectively:

$$W(x, x^i) = e^{-D(x, x^i)} \tag{9}$$

$$D(x_1, x_2) = \sum_{k=1}^{n} \left(\frac{x_{1k} - x_{2k}}{\sigma_k} \right)^2 \tag{10}$$

In Equation (10), each input variable has its own sigma value (σ_k). This formulation is different from Specht's[1] original work where he used a single sigma value for all input variables. However, using multiple sigmas can dramatically improve the performance of the GRNN, as explained by Masters[2].

Figure 1 shows a schematic depiction of the four layer GRNN. The first, or input layer, stores an input vector x. The second is the pattern layer which computes the distances $D(x, x^i)$ between the incoming pattern x and stored patterns x^i. The pattern nodes output the quantities $W(x, x^i)$. The third is the summation layer. This layer computes N_j, the sums of the products of $W(x, x^i)$ and the associated known output component y_i. The summation layer also has a node to compute D, the sum of all $W(x, x^i)$. Finally, the fourth layer divides N_j by D to produce the estimated output component y'_j that is a localized average of the stored output patterns.

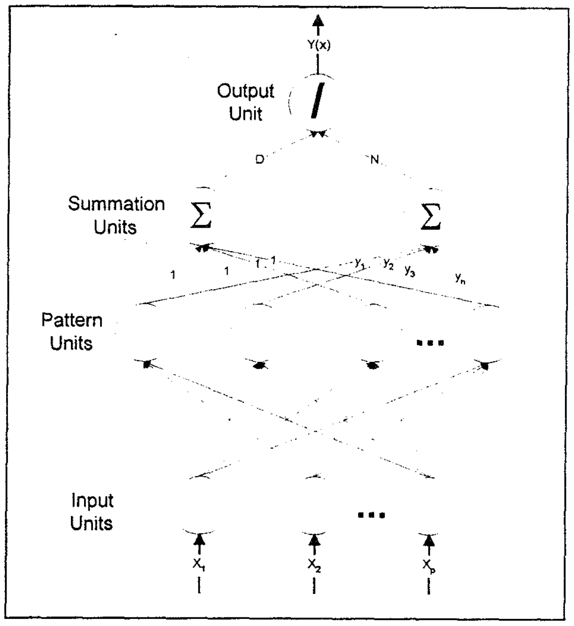

Figure 1) GRNN Diagram

4. NORMALIZED RBF DEFINITION

The definitions of the RBF and GRNN shown above were motivated by different ideas, but it is clear that the hidden layer processing is identical for both networks. This raises two questions. Why shouldn't the pseudo-inverse method be used to set the weights of the GRNN network? And why shouldn't the denominator term of the GRNN be included in the RBF network? We decided to integrate the two ideas into a new network. We call this new network the Normalized Radial Basis Function (NRBF) network. This name was chosen since the denominator term of the GRNN is equivalent to normalizing all the hidden layer output to sum to unity. The following derivation of the NRBF training algorithm will show how this is the case. The normalization constant for the NRBF network is the sum of all the hidden layer outputs:

$$D = \sum_{i=1}^{p} \phi(\|x - c_i\|) \tag{11}$$

This sum is then used to compute a scalar predicted output of the network as follows:

$$y' = \sum_{i=1}^{p} \lambda_i \frac{\phi(\|x - c_i\|)}{D} \tag{12}$$

Which is equivalent to:

$$y' = \frac{1}{D}\sum_{i=1}^{p} \lambda_i \phi(\|x - c_i\|) \tag{13}$$

As before, the output layer is a linear matrix operation and the output layer weights can be computed using the LMS algorithm. To derive the equation for the output layer weights it is easiest to examine the output layer equation in matrix form:

$$y' = A\lambda // D \tag{14}$$

where λ is a *p-by-m* matrix of weights, A is a *q-by-p* matrix of pattern layer outputs for all training patterns, y' is a *q-by-m* matrix of network outputs, and **D** is a *q-by-1* matrix consisting of the sum of each row in **A**. The matrix operation // is division of each element in a matrix by the associated element in a column vector.

Equation (6) once again expresses ideal performance, so we can expect optimal performance by setting the matrix of output layer weights (λ) according to:

$$\lambda = A^{+}(y**D) \tag{15}$$

where the matrix operation ** is multiplication of each element in a matrix by the associated element in a column vector.

5. BASIC PROPERTIES OF THE GRNN

This section reports on a series of experiments to verify and demonstrate some of the known basic properties of the GRNN. The experiments reveal the effect of the sigma values on the performance of the GRNN.

We generated *251* data points <*x,y*> with $y = x$, uniformly distributed over the interval *[0,1]*. In the pattern layer of the GRNN we stored 5 centers, also evenly distributed over this interval, i.e. *<0, 0>*, *<0.25, 0.25>*, *<0.5, 0.5>*, *<0.75, 0.75>*, and *<1, 1>*. The single sigma value was varied between 0 and 1.

The results of this experiment are summarized in Table 1. It can be observed that the performance of the GRNN is poor for sigma values that are either too large or too small. The optimal performance is around 0.15.

Table 1) Effect of Sigma

Sigma	Training RMS Error
0.05	0.063
0.1	0.039
0.15	0.011
0.2	0.012
0.25	0.025
0.5	0.111
1	0.224

When the sigmas are too large, a detrimental effect that we call the 'edge effect' occurs. Figure 2, produced using a sigma of 0.25, is a nice demonstration of this effect. To understand the edge effect, recall that the predicted output of the GRNN is always a weighted average of the outputs associated with the patterns stored in the pattern layer. The prediction of outputs that are at either extreme of the interval will be pulled away from the true value of the output, since the averaging is one-sided.

For very small sigma's we obtained a different kind of effect. Figure 3 shows a typical run with a sigma that is too small (σ = 0.05). For such a small sigma, the pattern layer nodes' activation becomes much more localized and one pattern node tends to dominate. Hence the predicted outcome will be very similar to the output associated with this single dominant pattern. The averaging which would normally occur is eliminated, and the network simply recalls the most similar pattern. Therefore, the predictions will follow the kind of step function. We call the resulting effect the 'step effect'.

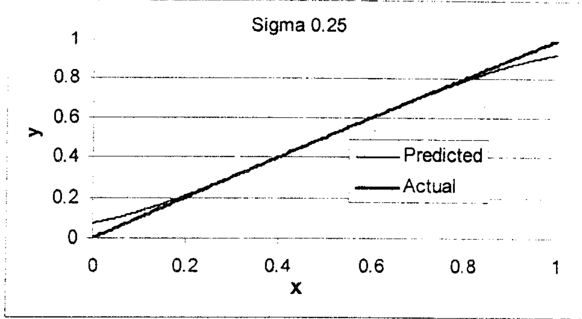

Figure 2) GRNN Output for Identity Mapping (σ = 0.25)

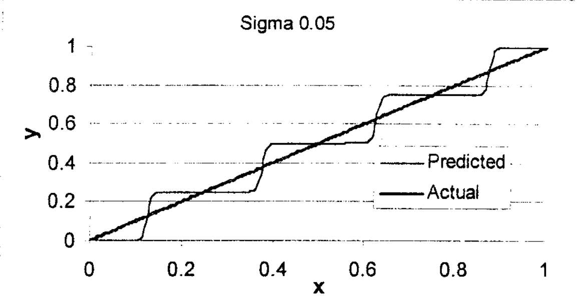

Figure 3) GRNN Output for Identity Mapping (σ = 0.05)

Figure 4 shows a run with near optimal sigma value, 0.15. The step effect and edge effect are both present yet they are barely visible. In general, we conclude that the optimal sigma is a compromise between the two effects. Decreasing sigma decreases the edge effect, but increases the step effect. The reverse relationship is also true.

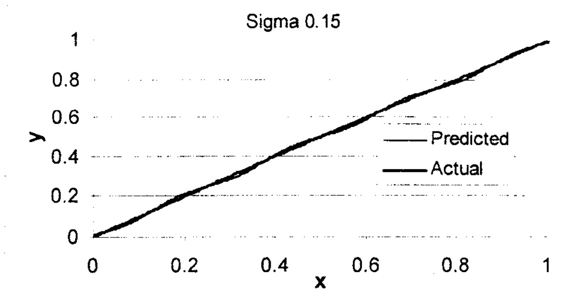

Figure 4) GRNN Output for Identity Mapping (σ = 0.15)

To investigate the effects of the number and position of the stored centers on GRNN performance in more detail, we used the same training data points from our first experiment. We then generated GRNN networks with *2, 5, 10, 50*, and *100* centers, all uniformly distributed over the interval. The error for each number of centers was computed for two cases: including the extreme values in the range and excluding the extreme values in the range. For example, when using *2* centers, we would use either *<0, 0>* and *<1, 1>*, or *<0.25, 0.25>* and *<0.75, 0.75>*. For any set of centers, we would determine the optimal sigma.

The outcome of this experiment is summarized in Table 2. We see that, as predicted, the RMS error decreases as the number of centers increases. Moreover, this reduction in the error is rather substantial. Table 2 also shows that including the extremes is always advantageous to the performance, even when only two centers are being used.

Table 2) Error versus Number of Centers

Number	Extremes included		Extremes not included	
Centers	Sigma	RMS Error	Sigma	RMS Error
2	0.637	0.0292	0.280	0.1071
5	0.169	0.0069	0.122	0.0285
10	0.078	0.0027	0.064	0.0106
50	0.015	0.0004	0.014	0.0012
100	0.007	0.0002	0.007	0.0005

A final observation from Table 2 is that as the number of centers increases, the optimal sigma value becomes smaller. This is because with more centers, the step effect will be reduced, while the edge effect remains the same. Hence the optimal compromise between the step effect and the edge effect will be obtained for a smaller sigma.

Given the above analysis, it is clear that an increase in the number of centers increases performance. On the other hand, the use of the pseudo-inverse method will also improve the performance of the GRNN. This time, the optimal sigma values will be larger, so that the step effect will be reduced. The pseudo-inverse method allows this to happen since the detrimental edge effect can be offset by adjusting the weights for pattern layer nodes near the edges of the input space. The results in the nest section will demonstrate this idea.

6. SIMPLE EXAMPLE

To demonstrate the effectiveness of this new method we will compare the performance of the Normalized RBF with that of the standard RBF and the GRNN for a simple example problem. The example problem is defined as follows:

$$y = \begin{cases} 2(x-0.5) & for \quad x > 0.5 \\ 2(0.5-x) & for \quad x < 0.5 \end{cases}$$

Training data was generated by selecting 101 points evenly distributed between $0.0 < x < 1$. GRNN, NRBF and RBF networks were trained by selecting 5 basis functions evenly distributed across the input range (0, 0.25, 0.5, 0.75, and 1.0) and using 3 values for σ (0.1, 0.15, and 0.2). Table 3 shows the optimum sigma values for each case and the RMS errors for the training data. Figure 5 shows the response of the three networks between $-0.2 < x < 1.2$.

Table 3 shows that as the number of centers increases the training error is significantly reduced, as expccted. This plot also shows that for our simple example the NRBF outperforms the other networks in terms of training error.

The RBF network output plots demonstrate that the RBF network extrapolation performance is poor and rapidly approaches zero. The NRBF network performs much better as its estimate is equal to the output of the nearest basis function.

Top right plot demonstrates the detrimental effect of not properly training an RBF network in the regions where $0.0<x<0.25$ and $0.75<x<1.0$. In this figure, the σ is too small and the centers are too widely distributed. Due to this problem, the RBF output approaches zero in the gaps between the basis functions, since the outputs of all hidden nodes are becoming very small. It does not interpolate well when the input vector is not close to any of the stored pattern centers. The NRBF network does not exhibit this problem since its outputs are normalized. It produces an output that is equal to the desired output of the nearest basis function or it produces an output which is an average between those basis functions to which it is the closest. With high sigma, this detrimental effect will decrease, however, it will never go away completely (for a small number of centers). We speculate that under normal circumstances, this should give the NRBF better interpolation performance that the RBF.

Figure 6 demonstrates the effect of various numbers of centers (hidden nodes). In all cases the centers are uniformly distributed and the optimum sigma was found. These plots show that the poor extrapolation of the RBF becomes worse as the concentration of centers increases. These plots also show that the NRBF exhibits good extrapolation behavior except for the largest number of hidden nodes.

7. SUMMARY AND CONCLUSIONS

The analysis and simple experimental results presented in this paper indicate that the NRBF network may be superior to the standard RBF network. We have demonstrated that at the very least, the NRBF network will have much better extrapolation capabilities than the standard RBF network. Further experimentation will be performed on real multi-input data to study the differences between the RBF and the NRBF networks additionally.

8. ACKNOWLEDGEMENTS

The authors would like to thank Timothy Masters for providing valuable suggestions. We also thank Cuong Dang, Elizabeth Dicke, and William Ford for their help on our experiments.

9. REFERENCES

[1] D. F. Specht, "A General Regression Neural Network", *IEEE Transactions on Neural Networks*, vol. 2, no. 6, pp. 568-576, November 1991

[2] T. Masters, *Advanced Algorithms for Neural Networks*, New York, NY: John Wiley and Sons, 1995

[3] D. S. Broomhead and D. Lowe, "Multi-variale Functional Interpolation and Adaptive Networks," *Complex Systems*, vol. 2, pp. 321-355. 1990.

Table 3) Training Error and Optimum Sigma

Number	GRNN		RBF		NRBF	
Centers	RMS	Sigma	RMS	Sigma	RMS	Sigma
5	0.0193	0.16	0.0191	0.17	0.0149	0.17
10	0.0236	0.07	0.0121	0.27	0.0106	0.18
20	0.0088	0.04	0.0032	0.12	0.0033	0.09
50	0.0024	0.014	0.0010	0.06	0.0010	0.05

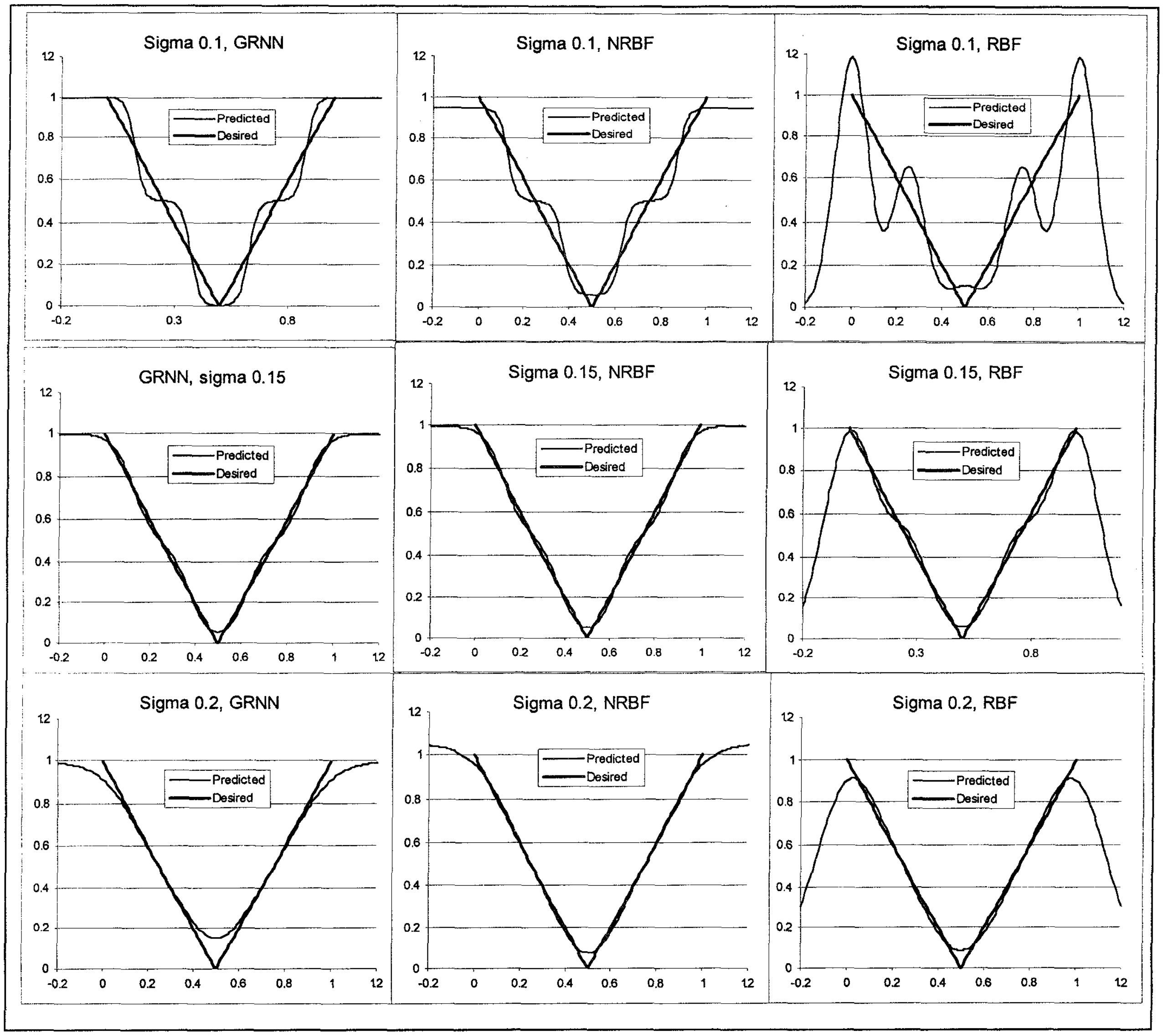

Figure 5) Network performance versus sigma with 5 centers. Column 1: GRNN, Column 2: NRBF, Column 3: RBF

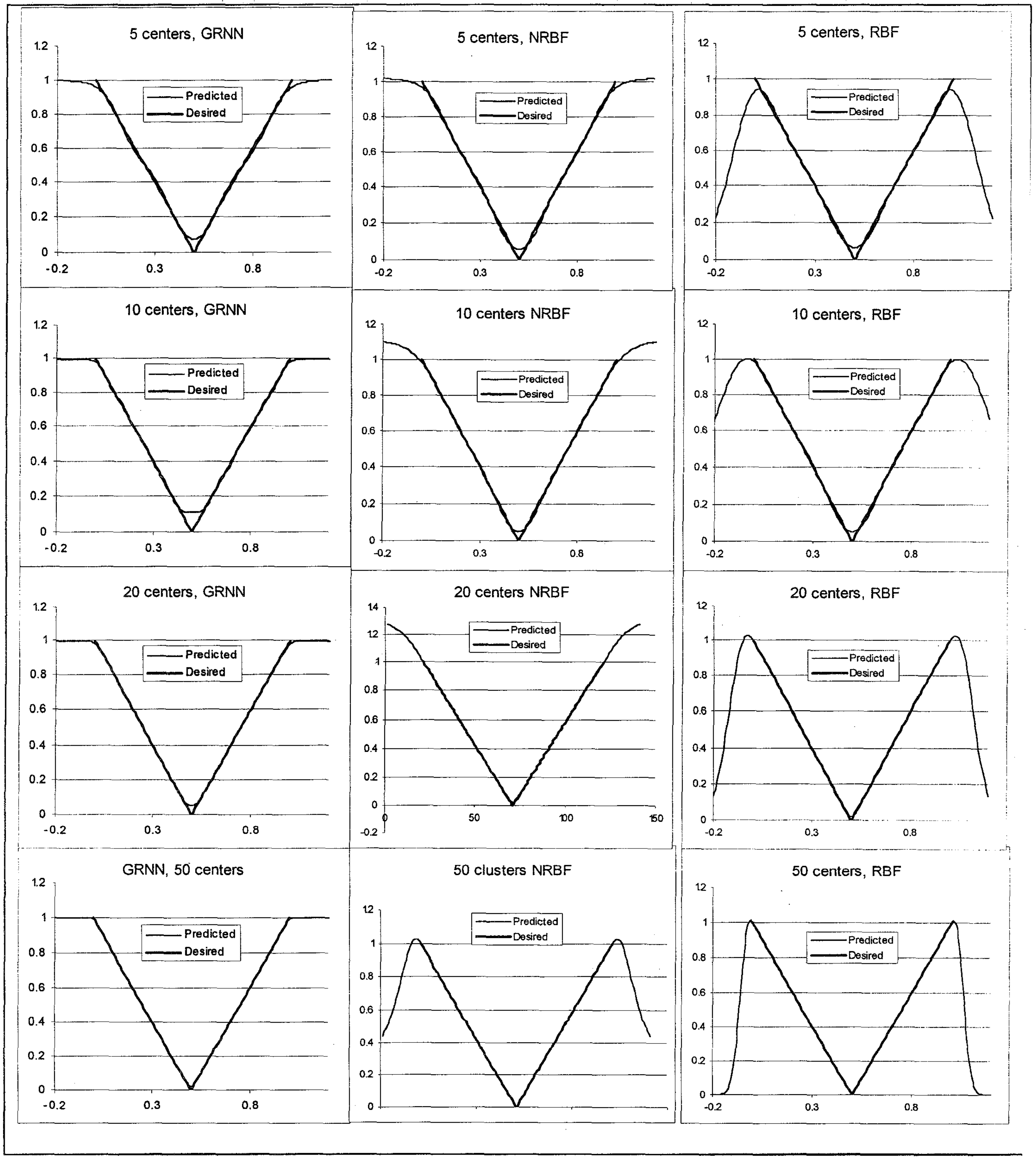

Figure 6) Network performance versus number of centers with optimum sigma. Column 1: GRNN, Column 2: NRBF, Column 3: RBF